07/872674

#4

SPECIFICATION

(Case 89,348)

Be it known that I, Michael F. Jones,

a United States citizen and a resident of

7 Foxglove Court, Nashua, New Hampshire 03062,

have invented a new and useful

ENCRYPTED DATA TRANSMISSION SYSTEM EMPLOYING MEANS FOR RANDOMLY ALTERING THE ENCRYPTION KEYS

as described in the attached specification:

BACKGROUND OF THE INVENTION

This invention relates to data transmission systems and, more particularly, to systems for transmitting enciphered data.

Data encryption provides security for transmitted data by scrambling the "clear text" data into "cipher text". Typically the transmitted data is scrambled in a manner selected by a unique key value (such as a 56-bit binary number) and unscrambled, at the receiving station, by a reverse process that requires that the same key value be known.

For increased data security, the encryption key value may be changed frequently to further reduce the likelihood that an unauthorized party may decipher the data. In such systems, new key values are sent at intervals from the transmitting station to the receiving station. The keys may be generated by a random number generator located at the transmitting end, encrypted in accordance with the currently active key, and transmitted along with the other data. At the receiving station, the encrypted key is extracted from the data stream, deciphered, and substituted at a designated time for the prior key. In such a system, if any of the transmitted keys are deciphered, the successive keys may be deciphered as well, so that all of the transmitted information may be decoded.

In accordance with a principle feature of the present invention, pseudo-random number generators are employed at both the transmitting and receiving stations to supply a like sequence of encryption keys to both the encryptor and decryptor, without these keys being transmitted in any form over the transmission facility. In accordance with the invention, to permit the two stations to communicate, each is

supplied in advance with a random number seed value which exclusively determines the numerical content of the sequence of numeric values generated by each of the two pseudo-random generators. In order that the two generators switch from one output key value to the next in synchronism, means are employed at both the transmitting and receiving stations to monitor the flow of transmitted data and to advance the random number generator each time the transmitted data satisfies a predetermined condition.

The monitoring function can advantageously be performed simply by counting the units of data being transmitted and by advancing each pseudo-random key generator each time the count reaches an agreed-upon interval number. In this way, no additional synchronization information needs to be added to the data stream. For even greater security, the interval number (which must be reached before the key is switched) may itself be a changing value generated by a random number generator, so that the duration during which a given key is active changes from key to key at times which are predictable only by the authorized recipient.

In accordance with still another feature of the invention, different random number seed values and different interval numbers (or different random number seed values for the generator of the interval numbers) may be associated with each of a plurality of remote locations with whom secured communication is required, so that the data on any given link is decipherable only by the authorized receiving station, even though other stations may have identical communication and decryption hardware.

As contemplated by still another feature of the invention, the encryption and decryption may advantageously be accomplished within a modem unit which also performs data compression and decompression, as well as error-handling functions. Advantageously, the compression, encryption and error-coding functions may all be performed (in that sequence) at the transmitting station by the same processor, while a like processor at the receiving end is suitably programmed to provide, in sequence, the error control, decryption, and decompression functions.

The principles of the invention may be applied to advantage in terminals connected as part of a secured communication network operating under central control. A key memory at each terminal may be loaded, by a secure communication from the central control, with encryption keys associated with other terminals with which secured communication is authorized. In this way, the central control can selectively permit or prohibit any terminal from decoding communications from any other terminal on a dynamically changing basis.

This and other features and advantages of the invention may be more clearly understood by considering the following detailed description of specific embodiments of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

In the course of the detailed description to follow, reference will frequently be made to the attached drawings, in which:

Figure 1 is a functional block diagram illustrating the basic signal processing steps which embody the invention;

Figure 2 is a hardware block diagram which shows a modem apparatus of the type contemplated by the invention; and

Figure 3 is a functional block diagram illustrating enhanced signal processing capabilities used in the preferred embodiment of the invention.

DESCRIPTION OF THE PREFERRED EMBODIMENT

Figure 1 - Basic Processing

Figure 1 illustrates the manner in which the data being transmitted is subjected to a sequence of signal processing steps as contemplated by the present invention. These processing steps are executed at a transmitting station 11 and at a receiving station 12 connected to opposite ends of a communications channel 13.

At the transmitting station 11, a source of data 15 supplies a serial data stream to the data input of an encryptor 17. The data from source 15 may take substantially any form, such as a file of text characters, each encoded as a 8-bit byte, or a file of numerical binary information expressed in 16-bit or 32-bit words. A block counter 21 monitors the stream of data from the source 15 and generates an "advance signal" each time the data meets a predetermined condition. Advantageously, the block counter 21 may simply count the number of bytes (characters), words or blocks of data being transmitted, compare the current count with a predetermined "interval number", and produce an advance signal each time the current count reaches the interval number (at which time the current count is reset to 0).

The advance signal produced by block counter 21 is supplied to the advance input of a pseudo-random number generator which supplies a sequence of encryption

key values to the key input of the encryptor 17. The content of the key sequence is predetermined by the combination of (1) the internal makeup of the generator 23 and by (2) a supplied random number seed value which initializes the generator. The generator 23 responds to each advance signal from block counter 21 by changing its output to the next successive output encryption key value. Thus, for example, the combination of counter 21 and generator 23 operate to change the encryption key each time total number of bytes transmitted is an exact multiple of the predetermined interval number.

The encryptor 17 translates fixed length segments of the data from source 15 ("clear text") into fixed-length "cipher text" output segments, each segment translation taking place in a manner uniquely determined by the encryption key currently supplied by the pseudo-random number generator 23. The encryptor 17 (and the decryptor 19, to be discussed) may advantageously employ the accepted NBIS Data Encryption Standard (DES), which codes and decodes data in 64-bit (8 byte) units in accordance with a 56-bit key. The block counter need not supply advance signals on boundaries between encryption units, nor does the generator 23 need to provide new key value precisely on encryption unit boundaries. Instead, the encryptor 17 may buffer the new keys temporarily, using it for the first time on the next successive encryption unit of data.

At the receiving station 12, the incoming cipher text is applied to the data input of the decryptor 31 whose key input is connected to receive a sequence of keys from the pseudo-random number generator 27. The clear text output from the decryptor 19 is applied to a data utilization device 33 and is monitored by a

block counter 29 which supplies advance signals to the number generator 27. Block counter 29 performs the identical function as that performed by the counter 21 at the transmitting station 11. and hence supplies advance signals to the generator 27 at precisely the same times (relative to the data stream) that counter 21 advances generator 23. Each time the current count reaches the interval number, the pseudo-random number generator 27 is advanced. Since the internal makeup of random number generator 27 is identical to that of generator 23, and since it is supplied with the same seed value, and since block counter 29 is supplied with the same interval number value as that supplied to the block counter 21, exactly the same sequence of keys will be supplied to the random number generators 23 and 27, and the keys will change at precisely the same time (relative to the data stream) to accurately decipher the transmitted data.

Of course, in order for the receiving station to successfully decipher the incoming cipher text, the receiving station 12 must by provided (in some fashion) with both the correct seed value and the correct interval number. These values are supplied to the receiving station in advance of the transmission by any secure means. However, once the receiver possesses these values, no further information is required to decipher the transmissions. No key values, key verification values, or key synchronization signals need accompany the transmitted ciphered text to control or coordinate the encryption or decryption processing, even though the encryption keys are continuously changing to enhance security.

Figure 2 - Hardware

The principles of the present invention may be advantageously implemented in a data communications modem having a hardware architecture of the type generally depicted in Figure 2 of the drawings. As shown, the modem operates under the supervisory control of a microprocessor 101 such as the model 80188 microprocessor available from Intel Corporation. The instructions and data operated on by the processor are stored in a memory subsystem 103 which is composed of both read-only memory (advantageously implemented as EPROM memory) and random access memory (RAM), Memory subsystem 103 is coupled to the microprocessor 101 by a memory address bus 105 and a data bus 107.

The data bus 107 also provides a data path to three peripheral devices: a display 109, a serial communications controller (SCC) 111, and a modem module 113. The SCC 111 may take the form of an integrated circuit such as the model 82530 controller manufactured by Intel Corporation. The modem module may be constructed using the model R9696 chip set available from Rockwell International Corporation, a cooperating set of integrated circuits capable of performing trellis-coded modulation and demodulation meeting the V.32 9600 baud communications protocol standard, as well as the V.22bis standard, and further includes analog/digital conversion circuits which provide an interface to a direct access adapter (DAA) 117. The adapter 117 may take the form of a type CH1818 integrated circuit DAA available from Cermetek Microelectronics, Inc.

The modem hardware shown in Figure 2 is used at both ends of the communications channel. At the transmitting end, data to be transmitted is supplied by the connected data terminal equipment (DTE) via the serial port 121 (e.g., a RS-

232c or RS-422 standard port). This asyncronous serial interface with the DTE typically operates under the combined control of the microprocessor 101 and the SCC 111 in accordance with a standard interface protocol (e.g., the V.42 standard protocol). The DTE (data terminal equipment) may be any terminal or computer adapted to communicate via this standard port using the selected serial protocol.

The encryption/decryption processing is essentially "transparent" to the DTE; that is, the data is enciphered and deciphered without effecting the content of the data sent by or received by the DTE. However, it is desireable to permit the connected DTE to send commands (such as extensions to the standard "AT command set") which will control encryption processing, turning encryption ON and OFF, and accepting seed values and interval numbers entered as "passwords" directly from the connected DTE.

Data signals from the DTE which are to be transmitted are encrypted as described above and shown in Figure 1, the random number seed values and the interval number values being pre-supplied to the microprocessor 101 and stored in memory subsystem 103. At the receiving end, the modem module 113 shown in Figure 2 receives the incoming data (typically as a 9600 baud trellis-coded signal adapted for transmission over the analog telephone link) and converts that incoming signal into data which is processed by microprocessor 101 and supplied via the SCC 111 to the connected DTE. In the receiving mode, microprocessor 101 decrypts the data as illustrated by the receiving station 12 in Figure 1.

Figure 3 - Enhancements

The principles of the invention may be advantageously employed to encipher and decipher data which is also compressed for enhanced transmission efficiency, and combined with error detection/correction coding. Moreover, the invention may utilize a key storage system to store unique keys for different called and calling parties, and may employ means for varying the interval number in a random fashion so that the time durations during which particular encryption keys are active varies in unpredictable ways. These further enhancements to the system are depicted in Figure 3 of the drawings which illustrates the preferred embodiment of the invention.

If the data signals are to be "compressed" for increased transmission efficiency (e.g., by Huffman encoding or the like), the compression processing of the data should precede encryption, because the encryption process inherently randomizes the data, eliminating the redundancy upon which efficient compression depends. On the other hand, error control processing (such as adding cyclic redundancy check (CRC) block checking codes) is best done after encryption in accordance with the invention, because successful synchronization of the advance signals from the block counters 21 and 29 requires substantially error-free data transmission (which the error-checking protocols insure).

As contemplated by the present invention, data compression, data encryption, and error control functions may all be performed by a single control processor. Thus, when a modem of the class shown in Figure 2 of the drawings is employed, the microprocessor 101 operates on the outgoing data stream by first performing data compression, then performing the encryption step, and finally performing the error detection/protection processing before forwarding the data on to the modem

module 113 for trellis coding and digital-to-analog conversion for transmission over the telephone network.

The signal processing functions used in this enhanced arrangement are shown in Figure 3 of the drawings. In Figure 3, the functional units employed in the basic system shown in Figure 1 are designated by the same numerals used in Figure 1, and the description of those units need not be repeated.

A data compressor 34 is shown connected between the data source 15 and the encryptor 17. In the hardware as seen in Figure 2, data compression may be conveniently performed by the microprocessor 101 on the data from the DTE obtained via the SCC 111. At the receiving station 12 as seen in Figure 3, a data decompressor 35 is connected between the decryptor 31 and the data utilization device 33. Note also that, as depicted In Figure 3, the data is monitored by the block counter 21 prior to compression, rather than afterwards. Correspondingly, at the receiving station 12, the block counter 29 monitors the data flow after it is decompressed. In this way, both counters monitor the same data stream. Both could be reconnected to monitor the compressed data stream if desired, however.

Error control processing is done by the error control coder 36 which, for example, might add cyclic redundancy check data to the data being transmitted to permit data correction in the error detector/corrector 37 at the receiving end (or to initiate a retransmission under the active error correction protocol. This error correction processing (at both ends) may be advantageously performed by the same microprocessor that performs the data compression and encryption functions.

To further enhance the security of the transmission, the duration of the interval during which each given key is active may be changed in a pseudo-random fashion. For this purpose, a pseudo-random number generator 38 is used at the transmitting station 11 to supply the interval numbers to the block counter 21. The generator 21 is advanced to a new number each time an advance signal is received from the output of block counter 21 over line 39 (so that a new interval number is supplied to the block counter 21 each time it advances the encryption key generator 23). Block counter may simply load the interval number generator 38 into an accumulator which is then decremented toward zero when it emits the advance signal to generator 23, at which time it is loaded with a new and different interval number from generator 38. At the receiving station 12, a pseudo-random generator 39 (which performs the same pseudo-random number generating process as the generator 38 at the transmitting station 11) supplies a sequence of interval numbers to counter 29. Generator 40 is advanced by the advance signals from counter 29 which also advance the encryption key generator 27.

The random number generators 23 and 38 at the transmitting station obtain their seed values from a key memory 50. Key memory 50 stores the random number keys indexed by destination (along with telephone dial-up numbers for automatic dialing). Similarly, at the receiving station, the seed values for the remote terminals from which the receiving station is authorized to receive information are stored in a key memory 60 connected to supply seed values to the generators 27 and 40. The key memories eliminates the need for authorized users to remember and enter keys before each transmission or reception.

In addition, the use of key memories allows the stations to be operate as terminals in a secure network under the control of a central station which, in separate transmissions over different secure links, enters (and erases) the keys needed by authorized sending and receiving stations connected to the network. In this way, the central station permits one network user to transmit to a single other user, or to "broadcast" to selected, authorized users on the network only, while enabling all terminals to use the network for unsecured transmissions.

Programming

The encryption and decryption operations may be performed by special purpose devices, such as those widely sold to implement the DES standard encryption method. As noted, however, the encryption function can be less expensively added by suitable programming of the microprocessor 101 to perform this function as well as the control, compression, and error handling functions. A working assembly language listing of a suitable DES encryption/decryption routine written for use with an 80188 microprocessor appears below. In the listing to follow, the function ENCRYPT.ASM is listed with comments, followed by an assembly language listing of the function KEY_SCHD.ASM, which calculates a sequence of 16 key-related values required in the DES algorithm. This sequence is pre-calculated when the DES key is changed to increase the speed of encryption and/or decryption.

```
title Data Block Encrypt/Decrypt Routines
;==============================================================================
;               (c) Copyright  Telequip Corp.  1989
;==============================================================================
;N a m e:
;
;       ENCRYPT.ASM
;
        name  ENCRYPT
;
;==============================================================================
;F u n c t i o n:
;
;       Performs DES encryption/decryption functions
;
;==============================================================================
;D e s c r i p t i o n:
;
;       This routine is designed to be called from a C or assembly language
; program to encrypt or decrypt a block of 8 data bytes.  Data bytes are copied
; from the source pointer and output is copied to the destination.  The C
; calling sequence is as follows:
;
;           void encrypt(dest, src, mode);
;
;               UNSC    *dest;      /* Near pointer to output destination   */
;               UNSC    *src;       /* Near pointer to input string         */
;               int     mode;       /* 1 <==> encipher, 0 <==> decipher     */
;                                   /* See MODEM.H for UNSC definition      */
;
;
; Segment naming conventions and the C calling sequence are those of
; Microsoft C 5.1.
;
;==============================================================================
include equs.inc

p   macro INREG, BIT, OREG
    test INREG, BIT
    lahf
    shl     AH, 1
    shl     AH, 1
    rcl     OREG, 1
    endm

;========================================
;       data and externals
;========================================

extrn   Key_s: byte
```

```
;======================================
;          procedure logic
;======================================

_TEXT   segment

        assume  CS: _TEXT, DS: DGROUP, SS: DGROUP

s_1     db  224,000,064,240, 208,112,016,064, 032,224,240,032, 176,208,128,016
        db  048,160,160,096, 096,192,192,176, 080,144,144,080, 000,048,112,128
        db  064,240,016,192, 224,128,128,032, 208,064,096,144, 032,016,176,112
        db  240,080,192,176, 144,048,112,224, 048,160,160,000, 080,096,000,208
s_2     db   15, 03, 01, 13,  08, 04, 14, 07,  06, 15, 11, 02,  03, 08, 04, 14
        db   09, 12, 07, 00,  02, 01, 13, 10,  12, 06, 00, 09,  05, 11, 10, 05
        db   00, 13, 14, 08,  07, 10, 11, 01,  10, 03, 04, 15,  13, 04, 01, 02
        db   05, 11, 08, 06,  12, 07, 06, 12,  09, 00, 03, 05,  02, 14, 15, 09
s_3     db  160,208,000,112, 144,000,224,144, 096,048,048,064, 240,096,080,160
        db  016,032,208,128, 192,080,112,224, 176,192,064,176, 032,240,128,016
        db  208,016,096,160, 064,208,144,000, 128,096,240,144, 048,128,000,112
        db  176,064,016,240, 032,224,192,048, 080,176,160,080, 224,032,112,192
s_4     db   07, 13, 13, 08,  14, 11, 03, 05,  00, 06, 06, 15,  09, 00, 10, 03
        db   01, 04, 02, 07,  08, 02, 05, 12,  11, 01, 12, 10,  04, 14, 15, 09
        db   10, 03, 06, 15,  09, 00, 00, 06,  12, 10, 11, 01,  07, 13, 13, 08
        db   15, 09, 01, 04,  03, 05, 14, 11,  05, 12, 02, 07,  08, 02, 04, 14
s_5     db  032,224,192,176, 064,032,016,192, 112,064,160,112, 176,208,096,016
        db  128,080,080,000, 048,240,240,160, 208,048,000,144, 224,128,144,096
        db  064,176,032,128, 016,192,176,112, 160,016,208,224, 112,032,128,208
        db  240,096,144,240, 192,000,080,144, 096,160,048,064, 000,080,224,048
s_6     db   12, 10, 01, 15,  10, 04, 15, 02,  09, 07, 02, 12,  06, 09, 08, 05
        db   00, 06, 13, 01,  03, 13, 04, 14,  14, 00, 07, 11,  05, 03, 11, 08
        db   09, 04, 14, 03,  15, 02, 05, 12,  02, 09, 08, 05,  12, 15, 03, 10
        db   07, 11, 00, 14,  04, 01, 10, 07,  01, 06, 13, 00,  11, 08, 06, 13
s_7     db  064,208,176,000, 032,176,224,112, 240,064,000,144, 128,016,208,160
        db  048,224,192,048, 144,080,112,192, 080,032,160,240, 096,128,016,096
        db  016,096,064,176, 176,208,208,128, 192,016,048,064, 112,160,224,112
        db  160,144,240,080, 096,000,128,240, 000,224,080,032, 144,048,032,192
s_8     db   13, 01, 02, 15,  08, 13, 04, 08,  06, 10, 15, 03,  11, 07, 01, 04
        db   10, 12, 09, 05,  03, 06, 14, 11,  05, 00, 00, 14,  12, 09, 07, 02
        db   07, 02, 11, 01,  04, 14, 01, 07,  09, 04, 12, 10,  14, 08, 02, 13
        db   00, 15, 06, 12,  10, 09, 13, 00,  15, 03, 03, 05,  05, 06, 08, 11

DEST        equ     word ptr [BP+4] ; Pointer to output buffer
SRC         equ     word ptr [BP+6] ; Pointer to input buffer
MODE        equ     word ptr [BP+8] ; Algorithm mode (1 = encrypt)
KEY_STRT    equ     word ptr [BP-2] ; Starting offset in key schedule
KEY_END     equ     word ptr [BP-4] ; Ending offset in key schedule

_encrypt    proc    near
            public  _encrypt        ; DES encryption algorithm

            push    BP
            mov     BP, SP          ; Standard C entry sequence
```

```
        sub     SP, 4              ; Stack space for locals
        push    SI
        push    DI
        mov     DI, SRC
        add     SRC, 8
        cmp     MODE, 1            ; Setup key schedule indices
        je      encd               ; Q. decipher
        mov     MODE, -8           ; Y, use schedule in reverse
        mov     KEY_STRT, offset DGROUP:Key_s + 120
        mov     KEY_END,  offset DGROUP:Key_s - 8
        jmp     ip
encd:   mov     MODE, 8            ; N, use schedule normally
        mov     KEY_STRT, offset DGROUP:Key_s
        mov     KEY_END,  offset DGROUP:Key_s + 128
ip:     mov     AX, [DI]           ; Initial permutation loop
        xchg    AL, AH
        xchg    SI, AX
        rcl     SI, 1
        rcr     CH, 1              ;   1
        rcl     SI, 1
        rcr     AH, 1              ;   2
        rcl     SI, 1
        rcr     CL, 1              ;   3
        rcl     SI, 1
        rcr     AL, 1              ;   4
        rcl     SI, 1
        rcr     DH, 1              ;   5
        rcl     SI, 1
        rcr     BH, 1              ;   6
        rcl     SI, 1
        rcr     DL, 1              ;   7
        rcl     SI, 1
        rcr     BL, 1              ;   8
        rcl     SI, 1
        rcr     CH, 1              ;   9
        rcl     SI, 1
        rcr     AH, 1              ;   10
        rcl     SI, 1
        rcr     CL, 1              ;   11
        rcl     SI, 1
        rcr     AL, 1              ;   12
        rcl     SI, 1
        rcr     DH, 1              ;   13
        rcl     SI, 1
        rcr     BH, 1              ;   14
        rcl     SI, 1
        rcr     DL, 1              ;   15
        rcl     SI, 1
        rcr     BL, 1              ;   16
        mov     SI, AX
        add     DI, 2              ;   index to next word in input
        cmp     DI, SRC            ;   Q. all 64 bits permuted
```

```
        jb      ip                  ;   N, continue in loop
                                    ; End of loop (approx. 400 '188 cycles)

        mov     DI, KEY_STRT
iter:   push    DX          ; AX:BX:CX:DX now with 1 to 64 (LR)
        push    CX
        push    BX
        push    AX
        sub     SI, SI              ; E - bit selection
                                    ;       high          low        carry
                                    ;    F EDCBA9  8  7 65432  1  0    C
                                    ;   -- ------ -- -- ----- -- -- ----
        shl     CX, 1               ;    2 ......  9 10 ..... 16  Z ( 1)
        rcl     DX, 1               ;   18 ...... 25 26 ..... 32  1 (17)
        adc     CX, SI              ;    2 ......  9 10 ..... 16 17 ( ?)
        mov     AX, CX              ;   AX: B2 B4
        mov     BX, DX              ;   BX: B6 B8
        shl     CX, 1               ;*   3 ...... 10 11 ..... 17  Z ( 2)
        rcl     DX, 1               ;   19 ...... 26 27 .....  1  2 (18)
        rcl     SI, 1               ;    Z ZZZZZZ  Z  Z ZZZZZ  Z 18 ( Z)
        shl     CX, 1               ;*   4 ...... 11 12 .....  Z  Z ( 3)
        rcl     DX, 1               ;   20 ...... 27 28 .....  2  3 (19)
        rcl     SI, 1               ;    Z ZZZZZZ  Z  Z ZZZZZ 18 19 ( Z)
        shl     CX, 1               ;*   5 ...... 12 13 ....Z  Z  Z ( 4)
        rcl     DX, 1               ;   21 ...... 28 29 .....  3  4 (20)
        rcl     SI, 1               ;    Z ZZZZZZ  Z  Z ZZZZ. 19 20 ( Z)
        shl     CX, 1               ;*   6 ...... 13 14 ...ZZ  Z  Z ( 5)
        rcl     DX, 1               ;   22 ...... 29 30 .....  4  5 (21)
        rcl     SI, 1               ;    Z ZZZZZZ  Z  Z ZZZ.. 20 21 ( Z)
        add     CX, SI              ;    6 ...... 13 14 ..... 20 21 ( Z)
        mov     SI, 03F3FH          ;   mask for 6 lsbs in each reg
        and     AX, SI              ;   AH: B2,   AL: B4
        and     CX, SI              ;   CH: B3,   CL: B5
        and     DX, SI              ;   DH: B7,   DL: B1
        and     SI, BX              ;   SI: B6        B8
                                    ; End of E - bit selection
                                    ; f(R, K)
        sub     BH, BH              ;   Clear msb of index
        mov     BL, AH              ;   B2  bits  5 -  8
        xor     BL, [DI+1]
        mov     AH, cs:s_2[BX]      ;                         (0Xh)
        mov     BL, DL              ;   B1         1 -  4
        xor     BL, [DI+0]
        add     AH, cs:s_1[BX]      ;                         (X0h)
        mov     BL, AL              ;   B4        13 - 16
        xor     BL, [DI+3]
        mov     AL, cs:s_4[BX]      ;                         (0Xh)
        mov     BL, CH              ;   B3         9 - 12
        xor     BL, [DI+2]
        add     AL, cs:s_3[BX]      ;                         (X0h)  1 - 16 in AX
        mov     BL, CL              ;   B5        17 - 20
        xor     BL, [DI+4]
```

```
mov      CH, cs:s_5[BX]   ;                              (X0h)
mov      BL, DH           ;   B7        25 - 28
xor      BL, [DI+6]
mov      CL, cs:s_7[BX]   ;                              (X0h)
mov      DX, SI
mov      BL, DH           ;   B6        21 - 24
xor      BL, [DI+5]
add      CH, cs:s_6[BX]   ;                              (0Xh)
mov      BL, DL           ;   B8        29 - 32
xor      BL, [DI+7]
add      CL, cs:s_8[BX]   ;                              (0Xh) 17 - 32 in CX
mov      BX, AX           ;   Permutation P follows
p   BL, 00000001B, SI     ;   16
p   BH, 00000010B, SI     ;    7
p   CH, 00010000B, SI     ;   20
p   CH, 00001000B, SI     ;   21
p   CL, 00001000B, SI     ;   29
p   BL, 00010000B, SI     ;   12
p   CL, 00010000B, SI     ;   28
p   CH, 10000000B, SI     ;   17
p   BH, 10000000B, SI     ;    1
p   BL, 00000010B, SI     ;   15
p   CH, 00000010B, SI     ;   23
p   CL, 01000000B, SI     ;   26
p   BH, 00001000B, SI     ;    5
p   CH, 01000000B, SI     ;   18
p   CL, 00000010B, SI     ;   31
p   BL, 01000000B, SI     ;   10
p   BH, 01000000B, DX     ;    2
p   BH, 00000001B, DX     ;    8
p   CH, 00000001B, DX     ;   24
p   BL, 00000100B, DX     ;   14
p   CL, 00000001B, DX     ;   32
p   CL, 00100000B, DX     ;   27
p   BH, 00100000B, DX     ;    3
p   BL, 10000000B, DX     ;    9
p   CH, 00100000B, DX     ;   19
p   BL, 00001000B, DX     ;   13
p   CL, 00000100B, DX     ;   30
p   BH, 00000100B, DX     ;    6
p   CH, 00000100B, DX     ;   22
p   BL, 00100000B, DX     ;   11
p   BH, 00010000B, DX     ;    4
p   CL, 10000000B, DX     ;   25
not      SI
not      DX               ;   End of permutation P
pop      CX               ; End of f(R, K)
xor      CX, SI
pop      AX
xor      DX, AX
pop      AX
pop      BX
```

```
        add     DI, MODE
        cmp     DI, KEY_END
        je      inv_ip
        jmp     iter                ; (approx. 16,720 '188 cycles for iter)
                                    ; 1 - 64 in AX:BX:CX:DX (LR)
inv_ip: mov     SI, AX              ; (33 - 48)
        mov     DI, CX              ; ( 1 - 16)
        push    BX                  ; (49 - 64)
        push    DX                  ; (17 - 32)
        mov     MODE, 4
                                    ; Inverse initial permutation loop
ipn:    shl     SI, 1               ;   33
        rcl     DL, 1
        shl     DI, 1               ;   1
        rcl     DL, 1
        shl     SI, 1               ;   34
        rcl     DH, 1
        shl     DI, 1               ;   2
        rcl     DH, 1
        shl     SI, 1               ;   35
        rcl     CL, 1
        shl     DI, 1               ;   3
        rcl     CL, 1
        shl     SI, 1               ;   36
        rcl     CH, 1
        shl     DI, 1               ;   4
        rcl     CH, 1
        shl     SI, 1               ;   37
        rcl     BL, 1
        shl     DI, 1               ;   5
        rcl     BL, 1
        shl     SI, 1               ;   38
        rcl     BH, 1
        shl     DI, 1               ;   6
        rcl     BH, 1
        shl     SI, 1               ;   39
        rcl     AL, 1
        shl     DI, 1               ;   7
        rcl     AL, 1
        shl     SI, 1               ;   40
        rcl     AH, 1
        shl     DI, 1               ;   8
        rcl     AH, 1
        dec     MODE
        jz      done
        cmp     MODE, 2
        jne     ipn
        pop     DI
        pop     SI
        jmp     ipn                 ; End inverse initial permutation loop
done:   mov     DI, DEST
        xchg    AL, AH
```

```
            xchg    BL, BH
            xchg    CL, CH
            xchg    DL, DH
            mov     [DI+0], AX      ; (approx. 420 clocks for ipn)
            mov     [DI+2], BX
            mov     [DI+4], CX
            mov     [DI+6], DX

            pop     DI
            pop     SI
            mov     SP, BP
            pop     BP
            ret

_encrypt    endp

_TEXT       ends

            end
;=============================================================================
;       KEY_SCHD.ASM
;
;       name  KEY_SCHD
;
;=============================================================================

;F u n c t i o n:
;
;       Calculate Sequence of 48-bit Ki from 56-bit Key
;

include equs.inc

ks  macro INREG, BIT
    test INREG, BIT
    lahf
    shl     AH, 1
    shl     AH, 1
    rcl     AL, 1
    endm

;=======================================
;       data and externals
;=======================================

_BSS    segment
public  Key_s

Key_s   db 128 dup(?)

_BSS    ends
```

```
;=======================================
;         procedure logic
;=======================================

_TEXT   segment

        assume  CS: _TEXT, DS: DGROUP

_key_schd   proc    near
            public  _key_schd        ; Compute key schedule

            push    BP
            mov     BP, SP
            push    SI
            push    DI
            mov     SI, [BP+4]   ; Offset of Key bits
            mov     AX, [SI]     ;  1 -  8  9 - 16 in AL AH
            mov     BX, [SI+2]   ; 17 - 24 25 - 32 in BL BH
            mov     CX, [SI+4]   ; 33 - 40 41 - 48 in CL CH
            mov     DX, [SI+6]   ; 49 - 56 57 - 64 in DL DH
            mov     DI, 4

    pc1:    shl     DH, 1        ; Permuted Choice 1 Loop
            rcl     SI, 1        ;   57  59  61  63
            shl     DL, 1
            rcl     SI, 1        ;   49  51  53  55
            shl     CH, 1
            rcl     SI, 1        ;   41  43  45  47
            shl     CL, 1
            rcl     SI, 1        ;   33  35  37  39
            shl     BH, 1
            rcl     SI, 1        ;   25  27  29  31
            shl     BL, 1
            rcl     SI, 1        ;   17  19  21  23
            shl     AH, 1
            rcl     SI, 1        ;    9  11  13  15
            shl     AL, 1
            rcl     SI, 1        ;    1   3   5   7
            dec     DI
            jz      e_pc1
            shl     DH, 1
            rcl     SI, 1        ;   58  60  62
            shl     DL, 1
            rcl     SI, 1        ;   50  52  54
            shl     CH, 1
            rcl     SI, 1        ;   42  44  46
            shl     CL, 1
            rcl     SI, 1        ;   34  36  38
            shl     BH, 1
            rcl     SI, 1        ;   26  28  30
            shl     BL, 1
            rcl     SI, 1        ;   18  20  22
```

```
        shl     AH, 1
        rcl     SI, 1           ;   10  12  14
        shl     AL, 1
        rcl     SI, 1           ;    2   4   6
        push    SI
        jmp     pc1
e_pc1:  mov     CX, SI
        mov     CH, CL          ; CH: 63 - 7
        pop     DX              ; DX: 61 - 5, 62 -  6
        mov     CL, DL          ; CX: 63 - 7, 62 -  6
        pop     BX              ; BX: 59 - 3, 60 -  4
        mov     DL, BL          ; DX: 61 - 5, 60 -  4
        shl     DL, 1
        shl     DL, 1
        shl     DL, 1
        shl     DL, 1           ; DX: 61 - 5, 28 -  4, 0, 0, 0, 0
        and     BL, 0F0H
        mov     SI, BX          ; SI: 59 - 3, 60 - 36, 0, 0, 0, 0
        pop     BX              ; BX: 57 - 1, 58 -  2
        sub     DI, DI          ; BX:SI - CX:DX

ks_lp:  shl     SI, 1           ; Rotate BX:SI
        rcl     BX, 1
        jnc     d1
        add     SI, 16
d1:     shl     DX, 1           ; Rotate CX:DX
        rcl     CX, 1
        jnc     c2
        add     DX, 16
c2:     cmp     DI, 0           ; Skip next rotate if reqd
        je      pc2
        cmp     DI, 8
        je      pc2
        cmp     DI, 64
        je      pc2
        cmp     DI, 120
        je      pc2
        shl     SI, 1           ; Rotate BX:SI
        rcl     BX, 1
        jnc     d2
        add     SI, 16
d2:     shl     DX, 1           ; Rotate CX:DX
        rcl     CX, 1
        jnc     pc2
        add     DX, 16
                                ;  1 -  8  9 - 16 in BX
                                ; 17 - 24 25 - 28 in SI
pc2:    mov     AL, 3
        ks      BX, 0004H       ; 14
        ks      SI, 8000H       ; 17
        ks      BX, 0020H       ; 11
        ks      SI, 0100H       ; 24
```

```
ks      BX, 8000H   ;  1
ks      BX, 0800H   ;  5
not     AL
mov     Key_s[DI+0], AL
mov     AL, 3
ks      BX, 2000H   ;  3
ks      SI, 0010H   ; 28
ks      BX, 0002H   ; 15
ks      BX, 0400H   ;  6
ks      SI, 0800H   ; 21
ks      BX, 0040H   ; 10
not     AL
mov     Key_s[DI+1], AL
mov     AL, 3
ks      SI, 0200H   ; 23
ks      SI, 2000H   ; 19
ks      BX, 0010H   ; 12
ks      BX, 1000H   ;  4
ks      SI, 0040H   ; 26
ks      BX, 0100H   ;  8
not     AL
mov     Key_s[DI+2], AL
mov     AL, 3
ks      BX, 0001H   ; 16
ks      BX, 0200H   ;  7
ks      SI, 0020H   ; 27
ks      SI, 1000H   ; 20
ks      BX, 0008H   ; 13
ks      BX, 4000H   ;  2
not     AL
mov     Key_s[DI+3], AL

                    ; 29 - 36 37 - 44 in CX
mov     AL, 3       ; 45 - 52 53 - 56 in DX
ks      CX, 0008H   ; 41
ks      DX, 0100H   ; 52
ks      CX, 2000H   ; 31
ks      CX, 0080H   ; 37
ks      DX, 2000H   ; 47
ks      DX, 0020H   ; 55
not     AL
mov     Key_s[DI+4], AL
mov     AL, 3
ks      CX, 4000H   ; 30
ks      CX, 0010H   ; 40
ks      DX, 0200H   ; 51
ks      DX, 8000H   ; 45
ks      CX, 0800H   ; 33
ks      DX, 1000H   ; 48
not     AL
mov     Key_s[DI+5], AL
                    ; 29 - 36 37 - 44 in CX
```

```
          mov     AL, 3         ; 45 - 52 53 - 56 in DX
          ks      CX, 0001H     ; 44
          ks      DX, 0800H     ; 49
          ks      CX, 0020H     ; 39
          ks      DX, 0010H     ; 56
          ks      CX, 0400H     ; 34
          ks      DX, 0080H     ; 53
          not     AL
          mov     Key_s[DI+6], AL
          mov     AL, 3
          ks      DX, 4000H     ; 46
          ks      CX, 0004H     ; 42
          ks      DX, 0400H     ; 50
          ks      CX, 0100H     ; 36
          ks      CX, 8000H     ; 29
          ks      CX, 1000H     ; 32
          not     AL
          mov     Key_s[DI+7], AL
          add     DI, 8
          cmp     DI, 128
          jae     e_ks
          jmp     ks_lp

    e_ks: pop     DI
          pop     SI
          pop     BP
          ret

_key_schd endp

_TEXT     ends

          end
```